SKAT Package

Seunggeun (Shawn) Lee

May 17, 2017

1 Overview

SKAT package has functions to 1) test for associations between SNP sets and continuous/binary phenotypes with adjusting for covariates and kinships and 2) to compute power/sample size for future studies.

2 Association test

An example dataset (SKAT.example) has a genotype matrix (Z) of 2000 individuals and 67 SNPs, vectors of continuous (y.c) and binary (y.b) phenotypes, and a covariates matrix (X).

```
> library(SKAT)
> data(SKAT.example)
> names(SKAT.example)

[1] "Z"   "X"   "y.c" "y.b"

> attach(SKAT.example)
```

To test for associations, SKAT_Null_Model function should be used in prior to run SKAT to estimate parameters under the null model of no associations.

```
> # continuous trait
> obj<-SKAT_Null_Model(y.c ~ X, out_type="C")
> SKAT(Z, obj)$p.value

[1] 0.002877041

> # dichotomous trait
> obj<-SKAT_Null_Model(y.b ~ X, out_type="D")
> SKAT(Z, obj)$p.value

[1] 0.1401991

>
```

When the trait is binary and the sample size is small, SKAT can produce conservative results. We developed a moment matching adjustment (MA) that adjusts the asymptotic null distribution by estimating empirical variance and kurtosis. By default, SKAT will conduct the MA adjustment when the sample size < 2000. In the following code, we use only 200 samples to run SKAT.

```
> IDX<-c(1:100,1001:1100)
> # With-adjustment
> obj.s<-SKAT_Null_Model(y.b[IDX] ~ X[IDX,],out_type="D")
```

```
Sample size (non-missing y and X) = 200, which is < 2000. The small sample adjustment is appl
```

```
> SKAT(Z[IDX,], obj.s, kernel = "linear.weighted")$p.value
```

```
[1] 0.1330844
```

```
>
```

If you don't want to use the adjustment, please set Adjustment=FALSE in the SKAT_Null_Model function.

```
> # Without-adjustment
> obj.s<-SKAT_Null_Model(y.b[IDX] ~ X[IDX,],out_type="D", Adjustment=FALSE)
> SKAT(Z[IDX,], obj.s, kernel = "linear.weighted")$p.value
```

```
[1] 0.147093
```

We recently developed an efficient resampling method to compute p-values for binary traits, and the methods are implemented in SKATBinary function. When you use the SKATBinary function, Adjustment=TRUE in SKAT_Null_Model is not necessary. Implemented methods are 1) Efficient resampling (ER); 2) ER with adaptive resampling (ER.A); 3) Quantile adjusted moment matching (QA); 4) Moment matching adjustment (MA); 5) No adjustment (UA); and 6) Hybrid. "Hybrid" (default method) selects a method based on the total minor allele count (MAC), the number of individuals with minor alleles (m), and the degree of case-control imbalance. Detailed description of these methods can be found in the following reference:

Lee, S., Fuchsberger, C., Kim, S., Scott, L. (2016) An efficient resampling method for calibrating single and gene-based rare variant association analysis in case–control studies. *Biostatistics* (2016) 17 (1): 1-15.

```
> # default hybrid approach
> out<-SKATBinary(Z[IDX,], obj.s, kernel = "linear.weighted")
> out$p.value
```

```
[1] 0.147093
```

```
>
```

2.1 Assign weights for each SNP

It is assumed that rarer variants are more likely to be causal variants with large effect sizes. To incorporate this assumption, the linear weighted kernel uses a weighting scheme and is formulated as $ZWWZ'$, where Z is a genotype matrix, and $W = diag\{w_1, \ldots, w_m\}$ is a weight matrix. In the previous examples, we used the default beta(1,25) weight, $w_i = dbeta(p_i, 1, 25)$, where $dbeta$ is a beta density function, and p_i is a minor allele frequency (MAF) of SNP i. Different parameters for the beta weight can be used by changing weights.beta. For example, weight.beta=c(0.5,0.5) will use the Madsen and Browning weight.

```
> SKAT(Z, obj, kernel = "linear.weighted", weights.beta=c(0.5,0.5))$p.value
```

```
[1] 0.4931639
```

You can use your own weight vector by using the weights parameter. For the logistic weight, we provide a function to generate the weight.

```
> # Shape of the logistic weight
>
> MAF<-1:1000/1000
> W<-Get_Logistic_Weights_MAF(MAF, par1=0.07, par2=150)
> par(mfrow=c(1,2))
> plot(MAF,W,xlab="MAF",ylab="Weights",type="l")
> plot(MAF[1:100],W[1:100],xlab="MAF",ylab="Weights",type="l")
> par(mfrow=c(1,2))
> # Use logistic weight
> weights<-Get_Logistic_Weights(Z, par1=0.07, par2=150)
> SKAT(Z, obj, kernel = "linear.weighted", weights=weights)$p.value
```

```
[1] 0.3293643
```

2.2 SKAT-O: Combined Test of burden test and SKAT

A test statistic of the combined test is

$$Q_\rho = (1-\rho)Q_S + \rho Q_B,$$

where Q_S is a test statistic of SKAT, and Q_B is a score test statistic of the burden test. The ρ value can be specified by using the r.corr parameter (default: r.corr=0).

```
> #rho=0, SKAT
> SKAT(Z, obj, r.corr=0)$p.value
```

```
[1] 0.1401991
```

```
> #rho=0.9
> SKAT(Z, obj, r.corr=0.9)$p.value
```

```
[1] 0.06031026
```

```
> #rho=1, Burden test
> SKAT(Z, obj, r.corr=1)$p.value

[1] 0.06095529
```

If method="optimal.adj" or "SKATO" (both are equivalent), SKAT-O method will be performed, which computes p-values with eight different values of $\rho = (0, 0.1^2, 0.2^2, 0.3^2, 0.4^2, 0.5^2, 0.5, 1)$ and then uses the minimum p-value as a test statistic. If you want to use the original implementation of SKAT-O, use method="optimal", which uses eleven equally spaced ρ values from 0 to 1 as a grid of ρs. We recommend to use "SKATO" or "optimal.adj", since it has a better type I error control.

```
> #Optimal Test
> SKAT(Z, obj, method="SKATO")$p.value

[1] 0.1008976

>
```

2.3 Combined test of common and rare variants

It is possible that both common and rare variants are associated with phenotypes. To test for combined effects of common and rare variants, SKAT_CommonRare function can be used. The detailed description of the combined test can be found in the following reference:

Ionita-Laza, I., Lee, S., Makarov, V., Buxbaum, J. Lin, X. (2013). Sequence kernel association tests for the combined effect of rare and common variants. *AJHG*, 92(6):841-53.

```
> # Combined sum test (SKAT-C and Burden-C)
>
> SKAT_CommonRare(Z, obj)$p.value

[1] 0.2238025

> SKAT_CommonRare(Z, obj, r.corr.rare=1, r.corr.common=1 )$p.value

[1] 0.1546374

> # Adaptive test (SKAT-A and Burden-A)
>
> SKAT_CommonRare(Z, obj, method="A")$p.value

[1] 0.4372293

> SKAT_CommonRare(Z, obj, r.corr.rare=1, r.corr.common=1, method="A" )$p.value

[1] 0.1548059

>
```

2.4 Impute missing genotypes.

If there are missing genotypes, SKAT automatically imputes them based on Hardy-Weinberg equilibrium. You can choose from “bestguess”, “fixed” or “random”. The “bestguess” imputes missing genotypes as most likely values (0,1,2), the “fixed” imputes missing genotypes by assigning the mean genotype value (2p, p is the MAF) and the ”random” imputes missing genotypes by generating binomial(2,p) random variables. The default imputation method for the SKAT function is “fixed” and for the SKATBinary function is “bestguess”.

```
> # Assign missing
> Z1<-Z
> Z1[1,1:3]<-NA
> # bestguess imputation
> SKAT(Z1,obj,impute.method = "bestguess")$p.value

[1] 0.1401991

> # fixed imputation
> SKAT(Z1,obj,impute.method = "fixed")$p.value

[1] 0.1401982

> # random imputation
> SKAT(Z1,obj,impute.method = "random")$p.value

[1] 0.1401991

>
>
```

2.5 Resampling

SKAT package provides functions to carry out resampling method to compute empirical p-values and to control for family wise error rate. Two different resampling methods are implemented. “bootstrap” conducts a parametric bootstrap to resample residuals from H_0 with adjusting for covariates. When there is no covariate, “bootstrap” is equivalent to the permutation. “perturbation” perturbs the residuals by multiplying standard normal random variables. The default method is “bootstrap”. From ver 0.7, we do not provide the “perturbation” method.

```
> # parametric boostrap.
> obj<-SKAT_Null_Model(y.b ~ X, out_type="D", n.Resampling=5000,
+ type.Resampling="bootstrap")
> # SKAT p-value
> re<- SKAT(Z, obj, kernel = "linear.weighted")
> re$p.value        # SKAT p-value

[1] 0.1401991
```

```
> Get_Resampling_Pvalue(re)         # get resampling p-value

$p.value
[1] 0.1433713

$is_smaller
[1] FALSE

> detach(SKAT.example)
```

When there are many genes/SNP sets to test, resampling methods can be used to control family-wise error rate. Examples are provided in the next section.

2.6 Adjust for kinship

If related individuals exist in your data, you need to adjust for kinship. SKAT_NULL_emmaX function uses linear mixed model (EMMAX) to estimate the variance component, which will be subsequently used to adjust for kinship. For the kinship adjustment, SKAT_NULL_emmaX function should be used instead of SKAT_Null_Model.

```
> data(SKAT.fam.example)
> attach(SKAT.fam.example)
> # K: kinship matrix
> obj<-SKAT_NULL_emmaX(y ~ X, K=K)
> SKAT(Z, obj)$p.value

[1] 0.2123192

> # SKAT-O
> SKAT(Z, obj, method="SKATO")$p.value

[1] 0.352943

> detach(SKAT.fam.example)
```

2.7 X chromosome test

Since male has only one copy of X-chromosome, special care is needed to test for associations in X-chromosome. We have developed a method to test for X-chromosome in region based rare variant test with and without X-inactivation. To use it, you need to use SKAT_Null_Model_ChrX to fit the null model and SKAT_ChrX for association tests. Detailed description of association tests in X-chromosome can be found in the following reference:

Ma, C., Boehnke, M., Lee, S., the GoT2D Investigators (2015) Evaluating the Calibration and Power of Three Gene-based Association Tests of Rare Variants for the X Chromosome, *Genetic Epidemiology*, 39 (7): 499-508.

```
> data(SKAT.example.ChrX)
> attach(SKAT.example.ChrX)
> Z = SKAT.example.ChrX$Z
> ##############################################################
> #        Compute the P-value of SKAT
>
> # binary trait
> obj.x<-SKAT_Null_Model_ChrX(y ~ x1 +x2 + Gender, SexVar="Gender", out_type="D", data=SKAT.e
> # run SKAT-O
> SKAT_ChrX(Z, obj.x, method="SKATO")$p.value

[1] 0.9156629

> detach(SKAT.example.ChrX)
```

3 Plink Binary format files

For the genome-wide data analysis, plink binary format files can be used in SKAT. To use plink files, plink bed, bim and fam files, and your own setid file that contains information of SNP sets are needed. Example files can be found on the SKAT/MetaSKAT google group page.

```
> # To run this code, first download and unzip example files
>
> ##############################################
> #         Generate SSD file
>
> # Create the MW File
> File.Bed<-"./Example1.bed"
> File.Bim<-"./Example1.bim"
> File.Fam<-"./Example1.fam"
> File.SetID<-"./Example1.SetID"
> File.SSD<-"./Example1.SSD"
> File.Info<-"./Example1.SSD.info"
> # To use binary ped files, you have to generate SSD file first.
> # If you already have a SSD file, you do not need to call this function.
> Generate_SSD_SetID(File.Bed, File.Bim, File.Fam, File.SetID, File.SSD, File.Info)

Check duplicated SNPs in each SNP set
No duplicate
1000 Samples, 10 Sets, 984 Total SNPs
[1] "SSD and Info files are created!"
```

Now you can open SSD and Info file and run SKAT.

```
> FAM<-Read_Plink_FAM(File.Fam, Is.binary=FALSE)
> y<-FAM$Phenotype
> # To use a SSD file, please open it first. After finishing using it, you must close it.
>
> SSD.INFO<-Open_SSD(File.SSD, File.Info)

1000 Samples, 10 Sets, 984 Total SNPs
Open the SSD file

> # Number of samples
> SSD.INFO$nSample

[1] 1000

> # Number of Sets
> SSD.INFO$nSets

[1] 10

> obj<-SKAT_Null_Model(y ~ 1, out_type="C")

> out<-SKAT.SSD.All(SSD.INFO, obj)

> out

$results
     SetID    P.value N.Marker.All N.Marker.Test
1  GENE_01 0.77747880           94            94
2  GENE_02 0.06245208           84            84
3  GENE_03 0.38416582          108           108
4  GENE_04 0.46179268          101           101
5  GENE_05 0.18548863          103           103
6  GENE_06 0.93255760           94            94
7  GENE_07 0.18897220          104           104
8  GENE_08 0.73081683           96            96
9  GENE_09 0.67366458          100           100
10 GENE_10 0.40310682          100           100

$P.value.Resampling
NULL

attr(,"class")
[1] "SKAT_SSD_ALL"
```

If you have a plink covariate file, Read_Plink_FAM_Cov function can be used to read both FAM and covariate files.

```
> File.Cov<-"./Example1.Cov"
> FAM_Cov<-Read_Plink_FAM_Cov(File.Fam, File.Cov, Is.binary=FALSE)
> # First 5 rows
> FAM_Cov[1:5,]
```

```
     FID IID PID MID Sex Phenotype         X1 X2
1 FID454   1   0   0   1  0.679793  1.0297614  1
2 FID977   1   0   0   1  0.836566  0.1846235  1
3 FID462   1   0   0   1 -0.408388 -0.6141158  1
4 FID958   1   0   0   1 -0.522305 -2.0226759  0
5 FID668   1   0   0   1 -0.328300 -0.8213776  0

> # Run with covariates
> X1 = FAM_Cov$X1
> X2 = FAM_Cov$X2
> y<-FAM_Cov$Phenotype
> obj<-SKAT_Null_Model(y ~ X1 + X2, out_type="C")

> out<-SKAT.SSD.All(SSD.INFO, obj)

> out

$results
     SetID    P.value N.Marker.All N.Marker.Test
1  GENE_01 0.77771227           94            94
2  GENE_02 0.06157071           84            84
3  GENE_03 0.39818504          108           108
4  GENE_04 0.46548442          101           101
5  GENE_05 0.18981516          103           103
6  GENE_06 0.94073952           94            94
7  GENE_07 0.18779019          104           104
8  GENE_08 0.74559501           96            96
9  GENE_09 0.66573796          100           100
10 GENE_10 0.40204308          100           100

$P.value.Resampling
NULL

attr(,"class")
[1] "SKAT_SSD_ALL"
```

To use custom weight, you need to make a weight file and read it using "Read_SNP_WeightFile" function. The weight file should have two columns, SNP ID and weight values. The output object of "Read_SNP_WeightFile" can be used as a parameter in SKAT.SSD functions

```
> # Custom weight
> # File: Example1_Weight.txt
> obj.SNPWeight<-Read_SNP_WeightFile("./Example1_Weight.txt")

> out<-SKAT.SSD.All(SSD.INFO, obj, obj.SNPWeight=obj.SNPWeight)

> out
```

```
$results
     SetID    P.value N.Marker.All N.Marker.Test
1  GENE_01 0.58647860           94            94
2  GENE_02 0.03286684           84            84
3  GENE_03 0.25752493          108           108
4  GENE_04 0.18486050          101           101
5  GENE_05 0.43670123          103           103
6  GENE_06 0.98039703           94            94
7  GENE_07 0.12460640          104           104
8  GENE_08 0.78814493           96            96
9  GENE_09 0.80206141          100           100
10 GENE_10 0.34070404          100           100

$P.value.Resampling
NULL

attr(,"class")
[1] "SKAT_SSD_ALL"
```

The output object of SKAT.SSD.All has an output dataframe object "results". You can save it using write.table function.

```
> output.df = out$results
> write.table(output.df, file="./save.txt", col.names=TRUE, row.names=FALSE)
>
```

If more than one gene/SNP sets are to be tested, multiple test should be adjusted to control for family-wise error rate. It can be done by the bonferroni correction. If gene/SNP sets are correlated, however, this approach can be conservative. Alternatively, you can directly control family wise error rate (FWER) using the resampling method.

```
> obj<-SKAT_Null_Model(y ~ 1, out_type="C", n.Resampling=1000, type.Resampling="bootstrap")
> out<-SKAT.SSD.All(SSD.INFO, obj)

> # No gene is significant with controling FWER = 0.05
> Resampling_FWER(out,FWER=0.05)

$result
NULL

$n
[1] 0

$ID
NULL

> # 1 gene is significnat with controling FWER = 0.5
> Resampling_FWER(out,FWER=0.5)
```

```
$result
    SetID    P.value N.Marker.All N.Marker.Test
2 GENE_02 0.06245208           84            84

$n
[1] 1

$ID
[1] 2
```

"SKAT.SSD.OneSet" or "SKAT.SSD.OneSet_SetIndex" functions can be used to test for a single gene/SNP set. Alternatively, you can obtain a genotype matrix using "Get_Genotypes_SSD" function and then run SKAT.

```
> obj<-SKAT_Null_Model(y ~ 1, out_type="C")
> # test the second gene
> id<-2
> SetID<-SSD.INFO$SetInfo$SetID[id]
> SKAT.SSD.OneSet(SSD.INFO,SetID, obj)$p.value

[1] 0.06245208

> SKAT.SSD.OneSet_SetIndex(SSD.INFO,id, obj)$p.value

[1] 0.06245208

> # test the second gene with the logistic weight.
> Z<-Get_Genotypes_SSD(SSD.INFO, id)
> weights = Get_Logistic_Weights(Z, par1=0.07, par2=150)
> SKAT(Z, obj, weights=weights)$p.value

[1] 0.7227001

>
```

SKAT_CommonRare function also can be used with SSD files.

```
> # test all genes in SSD file
> obj<-SKAT_Null_Model(y ~ X1 + X2, out_type="C")
> out<-SKAT_CommonRare.SSD.All(SSD.INFO, obj)
>

> out

$results
     SetID    P.value         Q N.Marker.All N.Marker.Test N.Marker.Rare
1  GENE_01 0.70833804  8678.419           94            94             0
2  GENE_02 0.01961982 11572.715           84            84             0
```

```
3  GENE_03 0.53912934 10656.127          108           108             0
4  GENE_04 0.34134633 10780.365          101           101             0
5  GENE_05 0.20548007 11592.362          103           103             0
6  GENE_06 0.92017774  7539.066           94            94             0
7  GENE_07 0.24712642 11545.172          104           104             0
8  GENE_08 0.66303494  9014.899           96            96             0
9  GENE_09 0.66044604  9399.011          100           100             0
10 GENE_10 0.30882075 10635.298          100           100             0
   N.Marker.Common
1               94
2               84
3              108
4              101
5              103
6               94
7              104
8               96
9              100
10             100

$P.value.Resampling
NULL

attr(,"class")
[1] "SKAT_SSD_ALL"
```

After finishing to use SSD files, please close them.

```
> Close_SSD()
```

```
Close the opened SSD file: /private/var/folders/xd/f13v97k97dq91vdbg87smshc0000gn/T/RtmpQngCI
```

3.1 Plink Binary format files: SKATBinary

SKATBinary functions can also be used with plink formatted files. This section shows an example code. Example plink files can be found on the SKAT/MetaSKAT google group page.

```
> # File names
> File.Bed<-"./SKATBinary.example.bed"
> File.Bim<-"./SKATBinary.example.bim"
> File.Fam<-"./SKATBinary.example.fam"
> File.Cov<-"./SKATBinary.example.cov"
> File.SetID<-"./SKATBinary.example.SetID"
> File.SSD<-"./SKATBinary.example.SSD"
> File.Info<-"./SKATBinary.example.SSD.info"
> # Generate SSD file, and read fam and cov files
> # If you already have a SSD file, you do not need to call this function.
> Generate_SSD_SetID(File.Bed, File.Bim, File.Fam, File.SetID, File.SSD, File.Info)
```

```
Check duplicated SNPs in each SNP set
No duplicate
2000 Samples, 30 Sets, 340 Total SNPs
[1] "SSD and Info files are created!"

> FAM<-Read_Plink_FAM_Cov(File.Fam, File.Cov, Is.binary=TRUE, cov_header=FALSE)
> # open SSD files
>
> SSD.INFO<-Open_SSD(File.SSD, File.Info)

2000 Samples, 30 Sets, 340 Total SNPs
Open the SSD file

> # No adjustment is needed
> obj<-SKAT_Null_Model(Phenotype ~ COV1 + COV2, out_type="D", data=FAM, Adjustment=FALSE)

> # SKAT
> out.skat<-SKATBinary.SSD.All(SSD.INFO, obj, method="SKAT")
> # SKAT-O
> out.skato<-SKATBinary.SSD.All(SSD.INFO, obj, method="SKATO")

> # First 5 variant sets, SKAT
> out.skat$results[1:5,]

  SetID    P.value N.Marker.All N.Marker.Test MAC  m Method.bin          MAP
1     1 0.92753378           11            11  18 17         ER 2.512149e-07
2     2 0.24947578            2             2   3  3         ER 3.544808e-02
3     3 0.60706345            7             7  19 19         ER 3.312382e-08
4     4 0.08566388           11            11  19 18         ER 6.640864e-08
5     5 0.63625247            4             4  18 18         ER 2.721199e-07

>
```

The effective number of tests and QQ plots can be obtained using the minimum achievable p-values (MAP).

```
> # Effective number of test is smaller than 30 (number of variant sets)
> # Use SKAT results
> Get_EffectiveNumberTest(out.skat$results$MAP, alpha=0.05)

[1] 28

> # QQ plot
> QQPlot_Adj(out.skat$results$P.value, out.skat$results$MAP)
>
```

4 Power/Sample Size calculation.

4.1 Dataset

SKAT package provides a haplotype dataset (SKAT.haplotypes) which contains a haplotype matrix of 10,000 haplotypes over 200kb region (Haplotype), and a dataframe with information on each SNP. These haplotypes were simulated using a calibrated coalescent model (cosi) with mimicking linkage disequilibrium structure of European ancestry. If no haplotype data are available, this dataset can be used to compute power/sample size.

```
> data(SKAT.haplotypes)
> names(SKAT.haplotypes)

[1] "Haplotype" "SNPInfo"

> attach(SKAT.haplotypes)
```

4.2 Power/Sample Size calculation

The following example uses the haplotypes in SKAT.haplotypes with the following parameters.

1. Subregion length = 3k bp
2. Causal percent = 20%
3. Negative percent = 20%
4. For continuous traits, $\beta = c|log_{10}(MAF)|$ (BetaType = "Log") with $\beta = 2$ at MAF = 10^{-4}
5. For binary traits, $log(OR) = c|log_{10}(MAF)|$ (OR.Type = "Log") with OR = 2 at MAF = 10^{-4}, and 50% of samples are cases and 50% of samples are controls

```
> set.seed(500)
> out.c<-Power_Continuous(Haplotype,SNPInfo$CHROM_POS, SubRegion.Length=5000,
+ Causal.Percent= 20, N.Sim=10, MaxBeta=2,Negative.Percent=20)

[1] "10/10"

> out.b<-Power_Logistic(Haplotype,SNPInfo$CHROM_POS, SubRegion.Length=5000,
+ Causal.Percent= 20, N.Sim=10 ,MaxOR=7, Negative.Percent=20)

[1] "10/10"

> out.c

$Power
          0.01     0.001     1e-06
500  0.5601495 0.4507543 0.2745436
1000 0.6983510 0.6372979 0.4477310
1500 0.7393476 0.6978347 0.5840998
```

```
2000 0.7741144 0.7169529 0.6649380
2500 0.8041370 0.7386689 0.6938517
3000 0.8224103 0.7660432 0.6997755
3500 0.8349515 0.7896737 0.7015918
4000 0.8484832 0.8037123 0.7049269
4500 0.8647970 0.8109526 0.7122846
5000 0.8834324 0.8165985 0.7253563

$R.sq
[1] 0.0693529

attr(,"class")
[1] "SKAT_Power"

> out.b

$Power
          0.01     0.001     1e-06
500  0.3894872 0.2757429 0.1330505
1000 0.5888308 0.4573657 0.2436726
1500 0.7021843 0.5859396 0.3485361
2000 0.7763091 0.6650800 0.4668508
2500 0.8234240 0.7280271 0.5483447
3000 0.8516985 0.7775865 0.5943673
3500 0.8718116 0.8108489 0.6269605
4000 0.8899993 0.8317031 0.6603647
4500 0.9081573 0.8464714 0.6968862
5000 0.9262225 0.8594656 0.7324297

attr(,"class")
[1] "SKAT_Power"

> Get_RequiredSampleSize(out.c, Power=0.8)

$`alpha = 1.00e-02`
[1] 2431.102

$`alpha = 1.00e-03`
[1] 3867.782

$`alpha = 1.00e-06`
[1] "> 5000"

> Get_RequiredSampleSize(out.b, Power=0.8)

$`alpha = 1.00e-02`
[1] 2251.417
```

```
$`alpha = 1.00e-03`
[1] 3336.919

$`alpha = 1.00e-06`
[1] "> 5000"

>
```

In this example, N.Sim=10 was used to get the result quickly. When you run the power calculation, please increase it to more than 100. When BetaType = "Log" or OR.Type = "Log", the effect size of continuous trait and the log odds ratio of binary traits are $c|log_{10}(MAF)|$, where c is determined by Max_Beta or Max_OR. For example, $c = 2/4 = 0.5$ when the Max_Beta = 2. In this case, a causal variant with MAF=0.01 has $\beta = 1$. For binary traits, $c = log(7)/4 = 0.486$ with MAX_OR=7. And thus, a causal variant with MAF=0.01 has log OR = 0.972.

Power_Continuous_R or Power_Logistic_R functions can be used to compute power with with non-zero r.corr (ρ). Since these functions use slightly different method to compute power, power estimates from Power_Continuous_R and Power_Logistic_R can be slightly different from estimates from Power_Continuous and Power_Logistic even when r.corr=0. If you want to computer the power of SKAT-O by estimating the optimal r.corr, please use r.corr=2. The estimated optimal r.corr is

$$r.corr = p_1^2(2p_2 - 1)^2,$$

where p_1 is the proportion of nonzero βs, and p_2 is the proportion of negative (or positive) βs among the non-zero βs.

```
> set.seed(500)
> out.c<-Power_Continuous_R(Haplotype,SNPInfo$CHROM_POS, SubRegion.Length=5000,
+ Causal.Percent= 20, N.Sim=10, MaxBeta=2,Negative.Percent=20, r.corr=2)

[1] "10/10"

> out.c

$Power
          0.01     0.001     1e-06
500  0.5584048 0.4465557 0.2700370
1000 0.6980094 0.6374870 0.4403217
1500 0.7367947 0.6977547 0.5830013
2000 0.7707641 0.7148115 0.6664808
2500 0.8032711 0.7341910 0.6946357
3000 0.8253110 0.7606592 0.6998229
3500 0.8407660 0.7863270 0.7011542
4000 0.8569269 0.8038311 0.7035340
4500 0.8759197 0.8137950 0.7089662
5000 0.8968032 0.8214246 0.7192218
```

```
$R.sq
[1] 0.0693529

$r.corr
[1] 0.0144

attr(,"class")
[1] "SKAT_Power"

> Get_RequiredSampleSize(out.c, Power=0.8)

$`alpha = 1.00e-02`
[1] 2449.686

$`alpha = 1.00e-03`
[1] 3890.566

$`alpha = 1.00e-06`
[1] "> 5000"

>
```